VŠB-Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2012 Miroslav Valečko

VŠB-Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webové GUI pro minimalizaci logických funkcí pomocí Karnaughových map
Web GUI for Minimising of Logical Functions Using Karnaugh Maps

2012 Miroslav Valečko

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Miroslav Valečko**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webové GUI pro minimalizaci logických funkcí pomocí Karnaughových map
Web GUI for Minimising of Logical Functions Using Karnaugh Maps

Zásady pro vypracování:

Cílem práce je realizace webové aplikace pro minimalizaci logických funkcí pomocí Karnaughových map. Uživatel by měl mít možnost po zadání logické funkce pomocí myši či klávesnice vytvářet v Karnaughově mapě smyčky pro minimalizaci funkce. V případě potřeby se po stisku pomocného tlačítka nabídnou další možné smyčky. Výstupem bude minimalizovaná funkce v součtovém a součinovém tvaru.

1. Popis algoritmu minimalizace logických funkcí pomocí Karnaughových map.
2. Návrh a realizace (implementace) webového grafického uživatelského rozhraní (GUI) pro minimalizaci logických funkcí pomocí Karnaughových map.
3. Vytvoření návodu pro práci s GUI.

Seznam doporučené odborné literatury:

[1] DIVIŠ, Zdeněk; CHMELÍKOVÁ, Zdeňka; ZDRÁLEK, Jaroslav. *Logické obvody*. 2. vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2008. 152 s. 978-80-248-1724-8.

[2] BERNARD, Jean Michel; HUGON, Jean; Le CORVEC, Robert. *Od logických obvodů k mikroprocesorům*. 2. nezm. vyd. Praha: SNTL - Nakladatelství technické literatury, 1988. 686 s.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum: 29. 4. 2012 ……

Podpis: Vaběko ……

Poděkování

Velice rád bych poděkoval svému vedoucímu, Ing. Jan Skapa, Ph.D., za cenné rady a odbornou pomoc, díky které jsem byl schopen vytvořit tuto bakalářskou práci. Dále pak bych chtěl poděkovat Ing. Michal Radecký, Ph.D., za připomínky k webové aplikaci a zvalidování kódu aplikace.

Abstrakt

Tato bakalářská práce se zabývá implementací webové aplikace pro minimalizování logických funkcí pomocí Karnaughových map. Nejprve si popíšeme co je to logická funkce. Řekneme si, jaká pravidla s logickými funkcemi platí v Booleově algebře. Dále se budu věnovat tomu, jak může být zadaná logická funkce. Seznámíme se s minimalizací logické funkce a třemi způsoby jak lze minimalizovat logickou funkci. Dalším krokem bude implementace kódu aplikace. V této aplikaci si může sám uživatel minimalizovat logickou funkci. Aplikace bude kontrolovat, zda minimalizaci provádí uživatel správně. Nebo uživatel ponechá minimalizaci na aplikaci.

Klíčová slova

Logická funkce, Booleova algebra, Karnaughova mapa, minimalizace, webová aplikace, Silverlight

Abstract

This bachelor work deals with implementation of web application for minimizing of logic functions using Karnaugh maps. First I will describe what a logic function is. We will study the rules that are applied in logical functions of Boolean algebra. After that, I will deal how logical functions can be specified. Afterwards the minimization of Boolean functions and three ways how to minimize the logical function will be explained. Another part will be implementation of the application code. In this application, a user can minimize a logical function on his/her own. The application will check if user gives logical function correctly to minimize, or he/she leaves let the application minimize itself.

Keywords

Logical function, Boolean algebra, Karnaugh map, minimize, web application, Silverlight

Seznam použitých zkratek a symbolů

A,B,C,D… – označení pro vstupní logické proměnné, velká písmena abecedy
f – funkce
$f_{(0)}$ – minimalizovaná funkce podle nul
$f_{(1)}$ – minimalizovaná funkce podle jedniček
D – disjunktivní zápis funkce
K – konjunktivní zápis funkce
V – vektorový zápis funkce
C# – C Sharp, programovací jazyk

Obsah

Úvod

Dnes se stále více setkáváme s číslicovou technikou, ať už si to uvědomíme nebo ne. Číslicová technika se nachází všude kolem nás, například v telekomunikacích, v počítačích, mobilních zařízeních, měřicí technice a dalších různých odvětvích.

S číslicovou technikou především také souvisí minimalizování logických funkcí. Díky minimalizaci logické funkce můžeme zjednodušit výsledné zapojení, ať se jedná o kombinační nebo sekvenční logické obvody. Výsledné zapojení je jednodušší a především méně nákladné na výrobu, proto je tedy velmi důležité rozumět problematice pro minimalizování logických funkcí.

Cílem této práce je vytvořit aplikaci pro minimalizování logických funkcí pomocí Karnaughovy mapy. Aplikaci budou moci používat uživatelé, kteří nejsou přímo znalí problematice minimalizování logických funkcí pomocí Karnaughovy mapy. Funkce se bude moci zadávat různými způsoby, například pomocí zaklikáním funkce do pravdivostní tabulky, disjunktivním, konjunktivním a vektorovým zápisem funkce nebo přímým zaklikáním do Karnaughovy mapy. Minimalizovat bude moci uživatel sám a aplikace bude kontrolovat, zda provádí minimalizaci správně podle pravidel nebo ponechá minimalizaci logické funkce na aplikaci.

Text této bakalářské práce je rozdělen do několika kapitol. Nejprve se seznámíme s logickou funkcí a Booleovou algebrou. Dále pak jakými způsoby můžeme zapsat logickou funkci. Poté se budeme zabývat třemi možnostmi, jak lze minimalizovat logickou funkci. V poslední kapitole se budu ve stručnosti věnovat implementaci aplikace.

1. Logická funkce

Logická funkce přesně definuje pro konečný počet vstupních proměnných, která vrací logické hodnoty. Vstupní proměnné, stejně jako logická funkce mohou nabývat pouze dvou hodnot a to logické 0 a logické 1.

Logickou funkci pro konečný počet vstupních proměnných lze zapsat v součtovém (úplná normální disjunktivní forma) a součinovém tvaru (úplná konjunktivní normální forma).

Základní součtový tvar funkce je dán součtem základních součinů přímých nebo negovaných proměnných [1], ve kterých funkce nabývá logické hodnoty 1.

Základní součinový tvar funkce je dán součinem dílčích součtů přímých nebo negovaných proměnných [1], ve kterých funkce nabývá logické hodnoty 0.

Tabulka 1: Kombinační tabulka

C	B	A	f(C,B,A)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Nyní zapíšeme funkci v součtovém tvaru, tedy pro ty řádky, ve kterých funkce nabývá logické hodnoty 1.

Tabulka 2: Dílčí součiny

C	B	A	Dílčí součiny (mintermy)
0	1	0	$\bar{C} * B * \bar{A}$
0	1	1	$\bar{C} * B * A$
1	0	1	$C * \bar{B} * A$
1	1	1	$C * B * A$

Základní součtový zápis vypadá takto:

$$f(C,B,A) = \bar{C} * \mathrm{B} * \bar{A} + \bar{C} * B * A + C * \bar{B} * A + C * B * A$$

Nyní zapíšeme funkci v součinovém tvaru, tedy pro ty řádky, ve kterých funkce nabývá logické hodnoty 0.

Tabulka 3: Dílčí součty

C	B	A	Dílčí součty (maxtermy)
0	0	0	$C + B + A$
0	0	1	$C + B + \bar{A}$
1	0	0	$\bar{C} + B + A$
1	1	0	$\bar{C} + \bar{B} + A$

Základní součinový zápis vypadá takto:

$$f(C, B, A) = (C + B + A) * (C + B + \bar{A}) * (\bar{C} + B + A) * (\bar{C} + \bar{B} + A)$$

2. Booleova algebra

V polovině 19. století vypracoval irský matematik G. Boole matematickou logiku jako aplikaci matematiky v oblasti logiky. Nazývá se algebra logiky nebo Booleova algebra. Je to souhrn pravidel a zákonů, které umožňují pracovat s logickými výroky jako s logickými proměnnými a funkcemi a to formou algebraických operaci [2].

Booleová algebra nabývá pouze dvou hodnot stejně jako logické proměnné a logické funkce i konstanty. Booleova algebra počítá tedy ve dvojkové soustavě. Nejčastěji se používají symboly **0** a **I** nebo **L** a **H**, jedná se tedy o logické symboly. Tyto symboly nám mohou například označovat dva stavy žárovky, která je zapojena v nějakém funkčním obvodu, kde logická **0** nám bude označovat stav žárovky, jako **nesvítí** a logická **I** nám bude označovat stav, jako **svítí**.

K vyjádření libovolné logické funkce se v Booleově algebře používají jen tři základní funkce. Je to logický součet, logický součin a logická negace. Těmito základními funkcemi můžeme vyjádřit libovolnou logickou operaci. Dále pomocí Booleovy algebry můžeme zjednodušovat logické funkce pomocí různých úprav a zákonů.

V číslicové technice se nejčastěji používají symboly **0** a **1**, kde podle Booleovy algebry se 0 rovná 0 a I rovná 1. Těmito symboly se zapisují konkrétní logické funkce, nebo stavy logických obvodů, ať se již jedná o kombinační logické obvody, sekvenční logické obvody, mikroprocesory atd.

2.1. Pravidla Booleovy algebry

Pravidla Booleovy algebry platí pouze pro logickou algebru, i když na první pohled se některá pravidla mohou shodovat s pravidly, které platí v číselné algebře. Například v Booleově algebře příklad $1 + 1 = 1$ ale v číselné algebře je $1 + 1 = 10$. Přehled pravidel Booleovy algebry jsou v tabulkách č. 4 a 5.

Tabulka 4: Základní vztahy Booleovy algebry

Logický součet	Logický součin	Negace
$0 + 0 = 0$	$0 * 0 = 0$	$\bar{0} = 1$
$0 + 1 = 1$	$0 * 1 = 0$	
$1 + 0 = 1$	$1 * 0 = 0$	$\bar{1} = 0$
$1 + 1 = 1$	$1 * 1 = 1$	

Tabulka 5: Přehled základních pravidel Booleovy algebry

Zákon	**Zápis**	
Asociativní	$(A+B)+C=A+(B+C)$	$(A*B)*C=A*(B*C)$
Komunikativní	$A+B=B+A$	$A*B=B*A$
Distributivní	$A+(B*C)=(A*B)+(A*C)$	$A*(B+C)=A*B+A*C$
Vyloučení třetího	$A+\bar{A}=1$	$A*\bar{A}=0$
Agresivnosti hodnot 0 a 1	$A+1=1$	$A*0=0$
Neutrálnosti hodnot 0 a 1	$A+0=A$	$A*1=A$
Absorpce	$A+A=A$	$A*A=A$
	$A+(A*B)=A*1+A*B=A*(1+B)=A$	$A*(A+B)=A*A+A*B=A*(1+B)=A$
Absorpce negace	$A*(\overline{\bar{A}+B})=A*B$	$A+\bar{A}*B=A+B$
	$\bar{A}*(A+B)=\bar{A}*B$	$\bar{A}+A*B=\bar{A}+B$
Dvojitá negace	$\bar{\bar{A}}=A$	
De Morganovy	$\overline{A+B}=\bar{A}*\bar{B}$	$\overline{A*B}=\bar{A}+\bar{B}$

3. Způsoby zápisu Booleovských funkcí

Na zápis Booleovské funkce nelze použít žádný známý způsob z matematiky. V číslicové technice však existuje několik různých variant, jak můžeme jednoznačně zapsat stejnou Booleovskou funkci.

3.1. Tabulkový zápis

Tabulkovým zápisem funkce respektive zápis pomocí pravdivostní tabulky funkce je jeden z nejpoužívanějších zápisů Booleovské funkce. Tento zápis je především vhodný pro zápis Booleovské funkce, která má maximálně 4 vstupní proměnné, neboť počet řádků vypočteme jako 2^n, kde n nám označuje počet vstupních proměnných. Počet řádků tedy získáme $2^4 = 16$. Samozřejmě že tento způsob zápisu můžeme použít i pro více proměnných, ale kdybychom měli sami vytvořit takovou tabulku kupříkladu pro 5 vstupních proměnných, museli bychom si nachystat tabulku o 32 řádcích, což by bylo pracné a nepřehledné. Na ukázku jak může vypadat takový zápis funkce, se podívejme na následující pravdivostní tabulky č. 6 a 7.

Tabulka 6: Úplný zápis funkce

Hodnota	C	B	A	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Tabulka 7: Zhuštěný zápis funkce

Hodnota	C	B	A	F
0,1	0	0	X	0
2,6	X	1	0	1
3	0	1	1	0
4,5	1	0	X	1
7	1	1	1	1

Pomocí zhuštěného zápisu funkce můžeme snížit počet řádků v tabulce, dosáhneme jej tak, že použijeme symbol X pro hodnoty vstupních proměnných. Tento symbol X znamená, že logická hodnota vstupní proměnné nabývá logických hodnot 0 a 1, přičemž výstupní logická hodnota zůstává stejná.

3.2. Neúplný zápis funkce

Logická funkce je neúplně zadaná, když pro některou nebo některé kombinace vstupních proměnných není funkce vůbec definována.

Tabulka 8: Neúplně zadaná funkce

Hodnota	B	A	f
0	0	0	0
1	0	1	1
2	1	0	1

3.3. Disjunktivní zápis funkce

Disjunktivní zápis funkce, je číselný zápis logické funkce. U tohoto zápisu se za symbolem D a v závorkách uvádějí řádky (indexy), v nichž logická hodnota funkce nabývá logické hodnoty 1, ostatní řádky mají logickou hodnotu 0.

$$f(C,B,A) = D(0,2,5)$$

Může nastat situace kdy je potřeba disjunktivním zápisem funkce zapsat funkci, která není úplná. Takového zápisu dosáhneme tak, že v zápisu funkce za symbolem D jsou opět závorky, které jsou vnější, a nejprve zapíšeme řádky, v nichž logická hodnota nabývá hodnot 1. Poté zapíšeme ještě jedny vnitřní závorky, kde jsou uvedeny řádky, ve kterých není logická hodnota definována, takže může nabývat logických hodnot 0 a 1. V takovémto řádku zapíšeme symbol X, ostatní řádky mají logickou hodnotu 0.

$$f(C,B,A) = D(0,2,5\ (1,4,7))$$

3.4. Konjunktivní zápis funkce

Konjunktivní zápis funkce, je také číselný zápis funkce. U tohoto zápisu se za symbolem K a v závorkách uvádí řádky, v nichž logická hodnota funkce nabývá logické hodnoty 0, ostatní řádky mají logickou hodnotu 1.

$$f(C,B,A) = K(1,3,6)$$

Opět může nastat situace jako u disjunktivního zápisu funkce a to že potřebujeme zadat neúplnou funkci. Postup je úplně totožný jako u disjunktivního zápisu, jenom se liší v tom, že uvádíme symbol K a řádky, v nichž logická hodnota nabývá hodnot 0. Dále uvádíme ve vnitřní závorce řádky, kde není funkce definována, a tedy může nabývat hodnot 0 a 1. V takovémto řádku zapíšeme symbol X, ostatní řádky mají logickou hodnotu 1.

$$f(C,B,A) = K(1,3,6\ (0,2))$$

3.5. Vektorový zápis funkce

Vektorový zápis funkce je nejčastěji zapsán v binární (dvojkové) soustavě. Kde poslední hodnota odpovídá nejnižší číselné hodnotě, tedy nule. Každý další index je o jeden vyšší a tedy hodnota nejvíce vlevo má nejvyšší hodnotu. Funkci, která je zapsána vektorovým zápisem, tak čteme zleva doprava a zapisujeme do tabulky sestupně.

$$f(C,B,A) = 01101101$$

Tabulka 9: Vektorový zápis funkce $f(C,B,A) = 01101101$

Hodnota	C	B	A	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Vektorovou funkci můžeme také zapsat v jiných soustavách, například v osmičkové, desítkové, šestnáctkové apod. Jestliže je funkce zadaná v jiné soustavě než dvojkové, nezbývá nám nic jiného než takovouto funkci převést do dvojkové soustavy. Viz. příklady níže, jedná se o stejnou funkci ale v jiných soustavách.

$$f(C,B,A) = 1110010_2$$
$$f(C,B,A) = 162_8$$
$$f(C,B,A) = 114_{10}$$
$$f(C,B,A) = 72_{16}$$

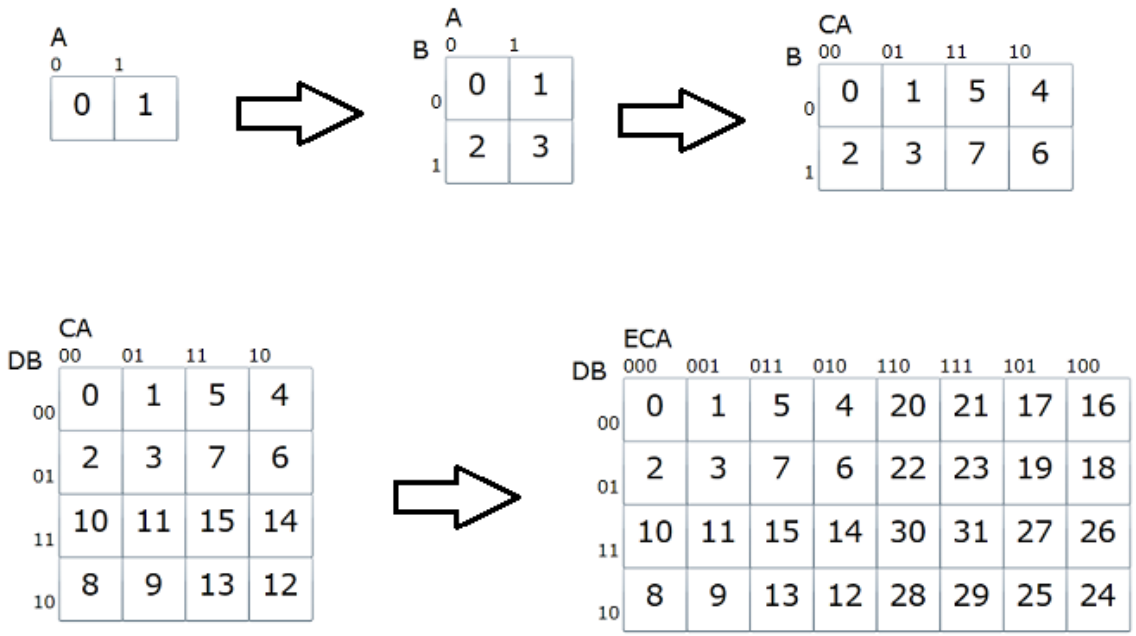
V případě rozšířené Booleovské funkce se může ve vektorovém zápisu objevit i symbol X.

$$f(C,B,A) = 1XX01X01$$

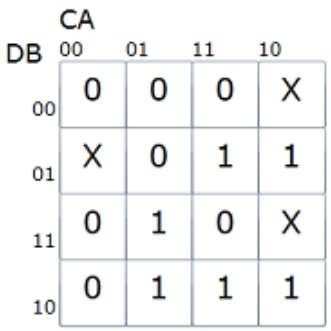
3.6. Zápis funkce pomocí grafické formy

Zápisem funkce pomocí grafické formy je funkce zapsaná v Karnaughově mapě. Karnaughova mapa se používá pro zápis logické funkce ale především také pro minimalizaci logických funkcí. Tato mapa se skládá z čtverečků, která tvoří síť obsahující 2^n čtverečků. Kde za písmeno n si můžeme dosadit počet vstupních proměnných dané funkce a vyjde nám velikost mapy, tedy počet čtverečků. Princip tvorby této mapy je založen na překlápění mapy při rozšiřování o další proměnnou. Neměli bychom nezapomenout na váhu proměnných

při vypočítávání hodnoty, která leží v konkrétním čtverečku. Jednotlivá políčka mohou nabývat pouze jedné hodnoty a to logické 0 a 1 popřípadě symbolem X.
Při tvorbě Karnaughovy mapy se využívá tzv. Grayův kód. Ten nám říká, že sousední čtverečky se liší maximálně v jedné proměnné. Podmínku splňují i čtverečky, které se nacházejí na krajích. Na příkladech níže je ukázán princip tvorby této mapy. V jednotlivých políčcích jsou pro ukázku zobrazeny hodnoty, které se v daném políčku nachází. Místo těchto hodnot je zde pak zapsaná funkce.



Obrázek 1: Ukázka tvorby Karnaughovy mapy



Obrázek 2: Ukázka zapsané funkce

4. Zjednodušování zápisu logické funkce

Zjednodušování zápisu logické funkce respektive minimalizace logické funkce je velice důležitá problematika v číslicové technice. Každou logickou funkci, která není minimalizována, můžeme zjednodušit. Snahou minimalizace logické funkce je snížit počet operací a také zmenšit složitost výsledného obvodu. Snížením počtu prvků v zapojení docílíme větší spolehlivosti dané funkce, také se zmenší především zpoždění zapojené funkce a hlavně snížíme náklady na výrobu a realizaci dané funkce. Minimalizovanou funkci pak zapisujeme v součtovém a součinovém tvaru. Proto je tedy důležité porozumět této problematice.

4.1. Algebraická minimalizace

Tato metoda minimalizace se používá především pro funkce, které mají maximálně 4 vstupní proměnné, poté se doporučuje provádět minimalizaci funkce pomocí jiné metody. U algebraické minimalizace využíváme intuice a zákonů Booleovy algebry. Snahou je dospět k co nejmenšímu zápisu výsledné funkce. Ukázkový příklad jak postupovat při Algebraické minimalizaci viz. níže.

Příklad 1: Zjednodušte funkci $f(C,B,A) = D(2,3,4,5)$ pomocí algebraické minimalizace v součtovém tvaru a v součinovém tvaru.

Tabulka 10: $\boldsymbol{f(C,B,A) = D(2,3,4,5)}$

Hodnota	C	B	A	**f**
0	0	0	0	**0**
1	0	0	1	**0**
2	0	1	0	**1**
3	0	1	1	**1**
4	1	0	0	**1**
5	1	0	1	**1**
6	1	1	0	**0**
7	1	1	1	**0**

Nejprve začneme tím, že podle pravdivostní tabulky si zapíšeme logickou funkci v úplném součtovém tvaru.

$$f_{(1)} = \bar{C} * B * \bar{A} + \bar{C} * B * A + C * \bar{B} * \bar{A} + C * \bar{B} * A$$

Nyní můžeme funkci zjednodušit pomocí algebraických úprav.

$$f_{(1)} = \bar{C} * B * (\bar{A} + A) + C * \bar{B} * (\bar{A} * A) = \bar{C} * B + C * \bar{B}$$

Výsledek v úplném **součtovém** tvaru.

$$\boldsymbol{f_{(1)} = \overline{C} * B + C * \overline{B}}$$

Dále můžeme podle pravdivostní tabulky zapsat logickou funkci v úplném součinovém tvaru.

$$f_{(0)} = (C + B + A) * (C + B + \bar{A}) * (\bar{C} + \bar{B} + A) * (\bar{C} + \bar{B} + \bar{A})$$

Zjednodušení funkce pomocí algebraických úprav, výsledek v úplném **součinovém** tvaru.

$$\boldsymbol{f_{(0)} = (C + B) * (\overline{C} + \overline{B})}$$

4.2. Minimalizace pomocí Mc-Cluskey metody

Minimalizace pomocí Mc-Cluskey metody se doporučuje používat, jestliže máme více než čtyři vstupní proměnné. Minimalizace spočívá v tom, že podle pravdivostní tabulky si vytvoříme další tabulku. V ní budeme mít pouze řádky, ve kterých výstupní logická hodnota funkce nabývá pouze stejné logické hodnoty, tedy buďto podle nul nebo podle jedniček. Poté v nové tabulce hledáme řádky, ve kterých se kombinace vstupních proměnných liší pouze u jedné proměnné. Takto projdeme všechny řádky a porovnáváme každý s každým. Jakmile najdeme řádky, ve kterých se liší pouze hodnota u jedné proměnné, můžeme tyto řádky sloučit a můžeme říct, že na této proměnné nezávisí. Řádky, které nesloučíme, nebo již nejdou dále sloučit, zapisujeme ve výsledné minimalizované funkci, tyto vektory nazýváme implikanty. Implikant může pokrýt současně více vektorů najednou. Na konci je nutné provést kontrolu pokrytí vektorů. Výslednou funkci zapisujeme buďto v součtovém, nebo součinovém tvaru a to, podle těch řádků, podle kterých jsme se rozhodli minimalizovat. Řádky ve kterých není funkce definována, se nemusí minimalizovat. Níže si uvedeme příklad, jak může vypadat postup minimalizace pomocí Mc-Cluskey metody.

Příklad 2: Proveďte minimalizaci funkce $f(D, C, B, A) = D(0,1,3,5,6,7,15)$ pomocí Mc-Cluckey metody a to jak v součtovém tvaru, tak v součinovém tvaru.

Tabulka 11: $f(D, C, B, A) = D(0, 1, 3, 5, 6, 7, 15)$

Hodnota	D	C	B	A	**f**
0	0	0	0	0	**1**
1	0	0	0	1	**1**
2	0	0	1	0	**0**
3	0	0	1	1	**1**
4	0	1	0	0	**0**
5	0	1	0	1	**1**
6	0	1	1	0	**1**
7	0	1	1	1	**1**
8	1	0	0	0	**0**
9	1	0	0	1	**0**
10	1	0	1	0	**0**
11	1	0	1	1	**0**
12	1	1	0	0	**0**
13	1	1	0	1	**0**
14	1	1	1	0	**0**
15	1	1	1	1	**1**

Nyní si vytvoříme tabulku a zapíšeme si řádky, v nichž se výstupní funkce rovná logické 1.

Tabulka 12: Tabulka pro porovnávání proměnných

Řádek	D	C	B	A
1	0	0	0	0
2	0	0	0	1
3	0	0	1	1
4	0	1	0	1
5	0	1	1	0
6	0	1	1	1
7	1	1	1	1

Vytvoříme si další tabulku, v nichž budou řádky, které slučujeme.

Tabulka 13: Sloučení řádků 1

Řádek	Sloučeno	D	C	B	A
1	1,2	0	0	0	-
2	2,3	0	0	-	1
3	2,4	0	-	0	1
4	3,6	0	-	1	1
5	4,6	0	1	-	1
6	5,6	0	1	1	-
7	6,7	-	1	1	1

Ještě můžeme sloučit dva řádky do jednoho, proto tedy opět vytvoříme další tabulku.

Tabulka 14: Sloučení řádků 2

Řádek	Sloučeno	D	C	B	A
1	2,5	0	-	-	1

Nyní máme tento postup hotový a můžeme se vrhnout na zápis výsledné funkce.

$$f_{(1)} = \overline{D} * \overline{C} * \overline{B} + \overline{D} * \overline{C} * A + \overline{D} * \overline{B} * A + \overline{D} * B * A + \overline{D} * C * A + \overline{D} * C * B + \overline{D} * A$$

Vektory se nám překrývají. V tabulce č. 12 máme celkem 7 řádků, jakmile se podíváme na tabulku č. 13 tak máme například řádky 2,3 a 4,6, které jsou ještě dále sloučené v tabulce č. 14 a proto nemusíme tyto řádky vypisovat při výsledném zápisu funkce. Stručněji řečeno máme 7 řádků, které máme očíslovány 1 až 7, a stačí nám ve výsledném minimálním zápisu funkce, když alespoň v jednom zápise bude dané číslo zastoupeno. Proto se tedy výsledný **součtový** minimální tvar skládá ze zápisu tabulky č. 13 s řádky 1,2; 5,6; 6,7 a tabulky č. 14.

$$\boldsymbol{f_{(1)} = \overline{D} * \overline{C} * \overline{B} + \overline{D} * C * B + C * B * A + \overline{D} * A}$$

Nyní budeme minimalizovat stejnou funkci jako na začátku s tím rozdílem, že budeme zapisovat řádky, v nichž logická hodnota výstupní funkce nabývá hodnoty logické 0. Postup bude úplně stejný, jako byl doposud, s tím rozdílem že výsledek bude zapsán v součinovém tvaru.

Tabulka 15: Tabulka pro porovnávání proměnných

Řádek	D	C	B	A
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0
4	1	0	0	1
5	1	0	1	0
6	1	0	1	1
7	1	1	0	0
8	1	1	0	1
9	1	1	1	0

Vytvoříme si tabulku, v nichž budou řádky, které slučujeme.

Tabulka 16: Sloučení řádků 1

Řádek	Sloučeno	D	C	B	A
1	1,5	-	0	1	0
2	2,7	-	1	0	0
3	3,4	1	0	0	-
4	3,5	1	0	-	0
5	3,7	1	-	0	0
6	4,6	1	0	-	1
7	4,8	1	-	0	1
8	5,6	1	0	1	-
9	5,9	1	-	1	0
10	7,8	1	1	0	-
11	7,9	1	1	-	0

Opět vytvoříme další tabulku, ve které budeme slučovat další řádky. Poslední tabulku vytvoříme proto, že v tabulce č. 17 se nám objevují stejné kombinace vstupních proměnných, proto je tedy sloučíme.

Tabulka 17: Sloučení řádků 2

Řádek	Sloučeno	D	C	B	A
1	3,8	1	0	-	-
2	3,10	1	-	0	-
3	4,6	1	0	-	-
4	4,11	1	-	-	0
5	5,7	1	-	0	-
6	5,9	1	-	-	0

Tabulka 18: Sloučení řádků 3

Řádek	Sloučeno	D	C	B	A
1	1,3	1	0	-	-
2	2,5	1	-	0	-
3	4,6	1	-	-	0

Nyní máme tento postup hotový a můžeme se vrhnout na zápis výsledné funkce.

$$f_{(0)} = (C + \bar{B} + A) * (\bar{C} + B + A) * (\bar{D} + C + B) * (\bar{D} + C + A) * (\bar{D} + B + A) * (\bar{D} + C + A) * (\bar{D} + B + \bar{A}) * (\bar{D} + C + \bar{B}) * (\bar{D} + \bar{B} + A) * (\bar{D} + \bar{C} + B) * (\bar{D} + \bar{C} + B) * (\bar{D} + \bar{C} + A) * (\bar{D} + C) * (\bar{D} + B) * (\bar{D} + A)$$

Vektory se nám opět překrývají, proto se stále nejedná o minimální zápis funkce. Aplikujeme tedy stejný postup, jak bylo vysvětleno výše. Výsledný **součinový** tvar.

$$\boldsymbol{f_{(0)} = (C + \bar{B} + A) * (\bar{C} + B + A) * (\bar{D} + C) * (\bar{D} + B) * (\bar{D} + A)}$$

4.3. Minimalizace pomocí Karnaughovy mapy

Popis a pravidla pro vytváření Karnaughovy mapy jsou popsána v kapitole 3.6. Minimalizace pomocí Karnaughovy mapy je grafická forma minimalizace logické funkce která je vhodná především pro více vstupních proměnných. Avšak je vhodná maximálně pro 5 vstupních proměnných, pokud minimalizujeme ve dvourozměrné matici. Samozřejmě můžeme minimalizovat ve dvourozměrné matici funkce, které mají více než 5 vstupních proměnných, ale již nedosáhneme minimálního zápisu funkce. Abychom dosáhli minimálního zápisu funkce pro více, než 5 vstupních proměnných museli bychom Karnaughovu mapu, tedy matici mít jako trojrozměrnou matici. Tato minimalizace spočítá v tom, že vytváříme smyčky v Karnaughově mapě, kde musí být pokryty všechny čtverečky, ve kterých se nachází stejná logická hodnota, tedy logická 1 nebo logická 0. Popřípadě můžeme kombinovat s nespecifikovaným stavem X pouze ale ve spojení s nějakou logickou hodnotou ať již to bude logická 1 nebo logická 0. Dalším důležitým pravidlem je, že smyčka musí být, co největší možná která lze vytvořit. Zároveň ale musí obsahovat počet políček, která jsou rovna 2^n kde je $n = 0,1,2,3, \ldots$ smyčka ve které se nachází pouze jedno políčko, je také smyčkou podle které se minimalizuje. Dále se snažíme dosáhnout co nejmenšího počtu smyček vytvořených v Karnaughově mapě. Jakmile máme vytvořenou smyčku, v zápisu minimalizované funkce zapisujeme pouze ty proměnné, které nemění svou logickou hodnotu ve vytvořené smyčce. Minimalizace se provádí podle jedniček, tak i podle nul, tedy v součtovém a součinovém tvaru. Opět si uvedeme příklad jak postupovat při minimalizaci pomocí Karnaughovy mapy.

Příklad 3: Proveďte minimalizaci funkce $f(D,C,B,A) = D(0,2,3(4,8,10,11,12,14))$ pomocí Karnaughovy mapy a to jak v součtovém tvaru, tak v součinovém tvaru.

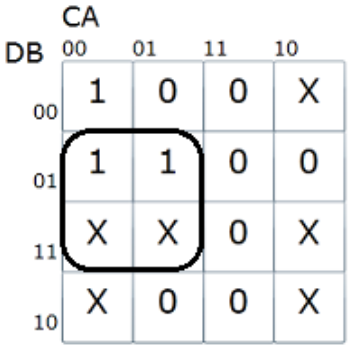
Tabulka 19: $f(D, C, B, A) = D(0, 2, 3(4, 8, 10, 11, 12, 14))$

Hodnota	D	C	B	A	**f**
0	0	0	0	0	**1**
1	0	0	0	1	**0**
2	0	0	1	0	**1**
3	0	0	1	1	**1**
4	0	1	0	0	**X**
5	0	1	0	1	**0**
6	0	1	1	0	**0**
7	0	1	1	1	**0**
8	1	0	0	0	**X**
9	1	0	0	1	**0**
10	1	0	1	0	**X**
11	1	0	1	1	**X**
12	1	1	0	0	**X**
13	1	1	0	1	**0**
14	1	1	1	0	**X**
15	1	1	1	1	**0**

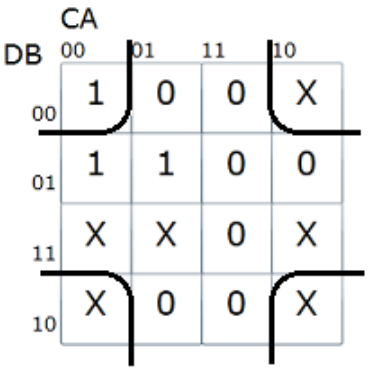
DB \ CA	00	01	11	10
00	1	0	0	X
01	1	1	0	0
11	X	X	0	X
10	X	0	0	X

Obrázek 3: Zapsaná funkce v Karnaughově mapě

Vytvořená smyčka nikdy nesmí vypadat jak písmeno L. Smyčka se dělá buď jako čtverec nebo jako obdélník, a to platí i když se vytvořená smyčka dělá přes jakékoliv okraje v Karnaughově mapě.



Obrázek 4: Smyčka č. 1

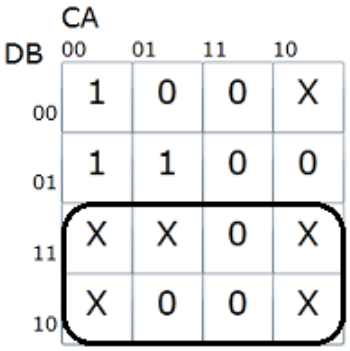


Obrázek 5: Smyčka č. 2

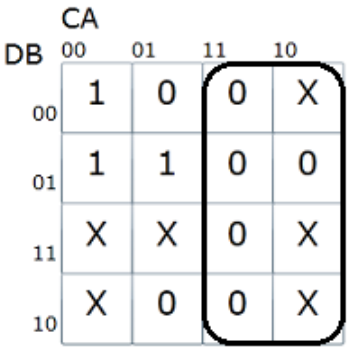
Zápis minimalizované funkce v **součtovém** tvaru, první součin odpovídá smyčce č. 1, druhý součin odpovídá smyčce č. 2.

$$\boldsymbol{f_{(1)} = \overline{C} * B + \overline{B} * \overline{A}}$$

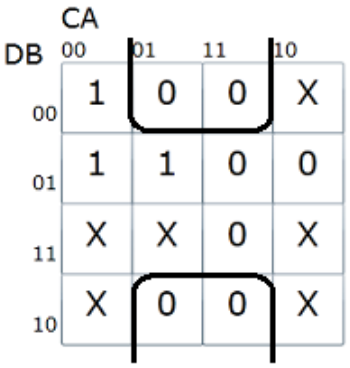
Nyní budeme provádět minimalizaci podle nul.



Obrázek 6: Smyčka č. 1



Obrázek 7: Smyčka č. 2



Obrázek 8: Smyčka č. 3

Zápis minimalizované funkce v **součinovém** tvaru, první součet odpovídá smyčce č. 1, druhý součet odpovídá smyčce č. 2 a třetí součet odpovídá smyčce č. 3.

$$\boldsymbol{f_{(0)} = (\overline{D}) * (\overline{C}) * (B + \overline{A})}$$

5. Implementace aplikace

Implementace webové aplikace probíhala v programovacím jazyce C# pro aplikační platformu Microsoft Silverlight ve verzi 4.0. Aplikaci popíšu velmi stručně, nebudu se věnovat každému řádku, neboť sepsaná aplikace má více než 6000 řádků. Podrobněji se budu věnovat části programu, která provádí minimalizaci logické funkce. Zde popíši pouze co, se vykonává při minimalizování logické funkce, respektive které metody se používají. Pro podrobnější náhled, jak jsou implementovány metody, doporučuji otevřít si vytvořenou aplikaci v nějakém vývojovém prostředí. Implementaci kódu aplikace jsem prováděl v Microsoft Visual Studio 2010 Ultimate.

5.1 Základní popis kódu aplikace

Nejprve se podíváme na diagram tříd, který jsem rozdělil na více obrázků, neboť samostatný diagram by se na jednu A4 stranu nevešel. U tříd není rozbaleno jaké proměnné a metody se zde používají, neboť by se u některých tříd nevešla tato třída na jednu stranu A4. Kód aplikace je rozdělen do pěti jmenných prostorů, ale diagramy budou na třech obrázcích. Mezi jednotlivými třídami neexistují žádné vazby, neboť v aplikaci je použito objektově orientované programování. Na obrázku č. 9 vidíme první diagram tříd, třída `MainPage` je v aplikaci hlavní třídou která tvoří hlavní stránku aplikace při jejím spuštění. U této třídy je staticky pouze nastavena barva pozadí a veškerý zbytek ovládacích prvků (popisky, tlačítka, pravdivostní tabulka atd.) jsou vytvořeny dynamicky, tedy již sepsaným algoritmem. Dynamický obsah ovládacích prvků jsem volil proto, že se neustále bude měnit počet vstupních proměnných, u kterých se pak bude měnit pravdivostní tabulka a Karnaughova mapa.

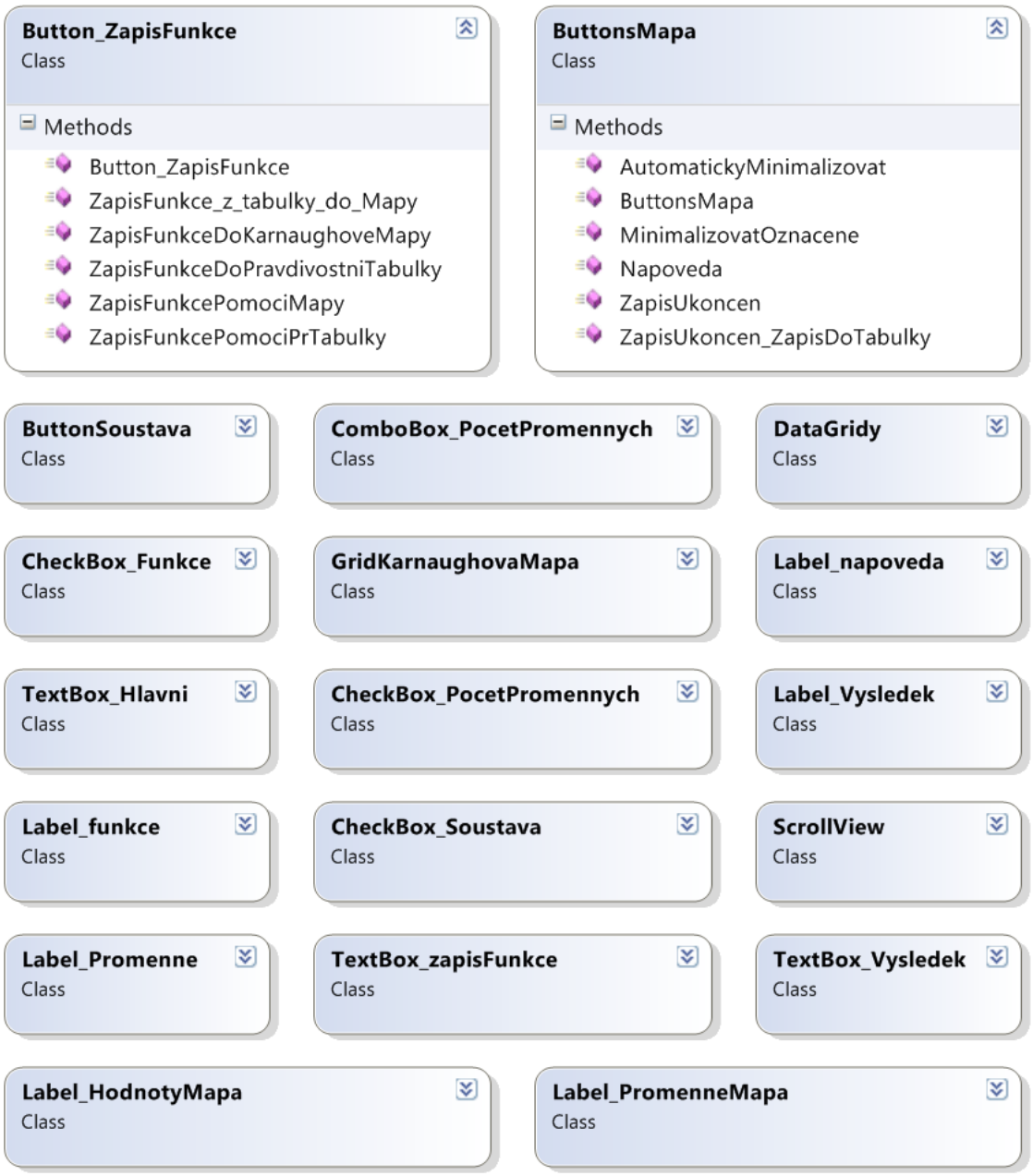
Obrázek 9: Diagram tříd 1

Pravdivostní tabulka je tvořena třemi tabulkami ačkoliv se tváří jako jedna velká. První tabulkou je, v nichž jsou hodnoty, tedy sloupec s názvem `Hodnota`. Další tabulka je reprezentována vstupními proměnnými, jsou to tedy sloupce s názvem `A,B,C….` Jako poslední tabulka je funkce, v nichž je název sloupce `Funkce`. Zde může uživatel zadávat funkci, pokud

zvolil tabulkový zápis funkce, nebo se sem zapíše funkce například, když zadal funkci pomocí disjunktivního zápisu funkce. Pro tyto tři tabulky mám napsané pomocné třídy, které reprezentují počet vstupních proměnných, a přes tyto třídy se tabulky plní daty. Na obrázku č. 10 si můžeme prohlédnout druhý diagram tříd, u nichž vidíme pomocné třídy až na třídu s názvem `TextNapovedy`.

Obrázek 10: Diagram tříd 2

Jak můžeme vidět u třídy `Bit_16`, třída má pouze čtyři vlastnosti (`properties`), které reprezentují vstupní proměnné pro 16 bitovou soustavu, metoda `Bit_16` je pouze konstruktorem této třídy. Ostatní třídy se pouze liší v názvu konstruktoru a počtu `properties`, které odpovídá počtu vstupním proměnným dané soustavy. Třídy `StavFunkce` a `Hodnoty` mají pouze jednu `property` a to `Funkce` a `Hodnota`. Ve třídě `TextNapovedy` jsou staticky napsány texty nápovědy, které se zobrazují během používání aplikace v horní části webového prohlížeče.
Poslední diagram tříd na obrázku č. 11 nám ukazuje třídy, ve kterých jsou naprogramovány ovládací prvky k aplikaci, které se zobrazují v aplikaci. Ovládací prvky se vytváří tak, že se vytvoří instance dané třídy, poté se zavolá metoda, která vrací požadovaný prvek, popřípadě se metodě předávají vstupní parametry. Například vytváření tlačítka nápověda, vytvoříme si instanci třídy `ButtonsMapa` a poté zavoláme metodu `Napoveda`. Ukázka metody `Napoveda`, která vytváří tlačítko nápověda je na obrázku č. 12.



Obrázek 101: Diagram tříd 3

Zavoláním metody `Napoveda`, která vrací objekt `Button`, musíme této metodě předat dva parametry. Prvním parametrem `marginX` zadáváme vytvářenému tlačítku, kde se bude nacházet v aplikaci od levého okraje. Druhým parametrem `marginY` se nastavuje u vytvářeného tlačítka, kde se bude nacházet od horní části obrazovky. Zadávaná vzdálenost se uvádí v počtech pixelů.

```
public Button Napoveda(double marginX, double marginY)
{
    Button b = new Button();
    b.Height = 38;
    b.Width = 500 / 3 - 6;
    b.HorizontalAlignment = HorizontalAlignment.Left;
    b.VerticalAlignment = VerticalAlignment.Top;
    b.Margin = new Thickness(marginX, marginY, 0, 0);
    b.Content = "Nápověda smyčky";
    return b;
}
```

Obrázek 11: Třída `ButtonsMapa`, metoda `Napoveda`

Jak vidíme na obrázku č. 12, tak si prvně vytvoříme novou instanci objektu `Button` (vytváří se tlačítko) která má proměnnou `b`. Poté nastavíme výšku a šířku tlačítka. Dále se nastavuje tlačítku zarovnání a to tedy doleva a nahoru. Posléze se nastaví pomocí `Margin` celkové umístění tlačítka v aplikaci. A jako poslední krok se nastaví popisek tohoto tlačítka. Tyto parametry se mohou tlačítku definovat v různém pořadí, nemusí to být tedy tak, jak to je na obrázku č. 12. Tato metoda se volá ze třídy `MainPage` a výsledný objekt tlačítka se ukládá ve volané třídě, kde se pomocí příkazu `grid.Children.Add(button_Napoveda);` přidá výsledné tlačítko na obrazovku. V této třídě je tomuto tlačítku nastavena událost a obslužná metoda, která bude zpracovávat uživatelskou akci pří kliknutí na toto tlačítko.
Třída `MainPage` patří k jedné z nejdůležitějších tříd, co se týče této aplikace. Z této třídy se volají veškeré ostatní třídy, ať už se jedná o vytváření ovládacích prvků či volání algoritmu samotné minimalizace.
Velice důležitou třídou je `MinimalizaceFunkce`. V této třídě se nachází veškerá logika aplikace pro minimalizace funkce pomocí Karnaughovy mapy. Ať se již jedná o kontrolu smyčky, zda je vytvořená správně, dále pak nápovědou smyčky, kterou vyvoláme zmáčknutím tlačítka nápověda, které se nachází pod Karnaughovou mapou, až po automatickou minimalizaci funkce.

5.2 Popis algoritmu minimalizace uživatelem

Při minimalizaci logické funkce, ať už uživatelem nebo samotnou aplikací, aplikace používá třídu `MinimalizaceFunkce`. Veškeré metody, které zde budou zmíněny, se nachází v této třídě. Pokud minimalizuje sám uživatel, vytvoří nějakou smyčku v Karnaughově mapě a poté zmáčkne tlačítko minimalizovat. Aplikace zkontroluje, zda vytvořená smyčka uživatelem je správně, nejprve se zkontroluje, zda funkce není cela minimalizována. Kontrola se provádí ve třídě `MinimalizaceFunkce` v metodě `Vse_minimalizovano` které se předává parametr `minimalizovana_hodnota`. Tato proměnná je typu `bool` a jedná se o dvourozměrnou matici, ve které si aplikace ukládá, které logické hodnoty již byly minimalizovány. Metoda `Vse_minimalizovano` má návratový typ `bool` a vrací nám hodnoty `true/false.` Poté víme, zda je či není logická funkce minimalizována celá. Metodu máme na obrázku č. 13.

```
public bool Vse_minimalozovano(bool[,] matice)
{
    for (int i = 0; i < matice.GetLength(0); i++)
    {
        for (int j = 0; j < matice.GetLength(1); j++)
        {
            if (!matice[i, j])
            {
                return false;
            }
        }
    }
    return true;
}
```

Obrázek 12: Trida `MinimalizaceFunkce`, metoda `Vse_Minimalizovano`

Jakmile metoda vrací hodnotu `false`, jsou poté volány metody `SpravnyPocetSmicek`, `LzeMinimalizovat`. Ty kontrolují, zda vytvořená smyčka odpovídá pravidlům minimalizace logické funkce pomocí Karnaughovy mapy, které jsou zmíněné v kapitole 4.3. Jakmile je vytvořená smyčka správně, zavolá se metoda `Zminimalizovano` která kontroluje, zda hodnoty které jsou ve smyčce, nejsou již minimalizovány. Pokud hodnoty nejsou minimalizovány, dojde k poslední kontrole, a to zda vytvořená smyčka obsahuje maximální počet políček pro minimalizaci. Jestliže je vše správně dojde k minimalizování smyčky a zápisu do příslušného text boxu, tedy podle nul nebo podle jedniček. Pokud ale smyčka není správně vytvořena, dojde k varovné události kdy je uživatel informován, o jakou chybu se jedná.

5.3 Popis algoritmu automatické minimalizace

Při automatické minimalizaci logické funkce se spouští ze třídy `MainPage` konkrétně metoda `button_AutomatickyMinimalizovat_Click`, která obsluhuje událost při kliknutí na tlačítko automaticky minimalizovat. V této metodě vytvářím instanci na třídu `MinimalizaceFunkce` a poté volám její metody pro automatickou minimalizaci. Nejprve si ale předám do volané třídy proměnné, aby tato třída mohla pracovat s logickou funkcí, která je v Karnaughově mapě. V cyklu `while` volám metodu `Vse_mininalizovano` která je nastavena v podmínce u cyklu. Jestliže volaná metoda vrací `false`, tedy že funkce není celá minimalizována tak vykoná příkazy uvnitř cyklu a to dokola dokud nebude celá logická funkce minimalizována. Část ukázky tohoto kódu se nachází na obrázku č. 14.

```
MinimalizaceFunkce minF = new MinimalizaceFunkce();
minF.hodnotyPozicMapy = hodnotyPozicMapy;
while (!minF.Vse_minimalozovano(minimalizovana_hodnota))
{
    minF.minimalizovana_hodnota = minimalizovana_hodnota;
    minF.LogHodnotyFunkce_v_mape = LogHodnotyFunkce_v_mape;
    minF.textBox_KarnMapa = textBox_KarnMapa;
    minF.binarniHodnoty = binarniHodnoty;
    minF.tabX = tabX;
    minF.AutomatickaMinimalizace();
    .
    .
    .
```

Obrázek 13: Metoda `button_AutomatickyMiniminalizovat_Click`, část kódu

Spuštěním metody `AutomatickaMinimalizace`, začne procházet aplikace Karnaughovu mapu a hledá logické hodnoty, které nejsou minimalizovány. Jakmile najde políčko, které není minimalizované, vytvoří smyčku. Takto vytvořená smyčka pak kontroluje opět pomocí cyklu `while` kde v podmínce je nastavena metoda `MaximalniSmycka` kdy se kontroluje, zda neexistuje větší možná smyčka. Jestliže tato metoda zjistí, že vytvořená smyčka není maximálně veliká, uloží pozici nové větší smyčky. A vrátí se zpátky do metody `AutomatickaMinimalizace`. Tato metoda zpracuje novou smyčku, opět si ji zapíše a znovu kontroluje, zda neexistuje větší smyčka. Jakmile aplikace našla maximálně velkou smyčku, zakreslí se do Kaurnaughovy mapy a algoritmus se vrací zpátky do třídy `MainPage`. Nakonec se takto vytvořená smyčka minimalizuje a zapíše do příslušného text boxu, tedy podle nul nebo podle jedniček. Tento proces se opakuje do doby, než nebudou všechny logické hodnoty minimalizovány.

Závěr

V této bakalářské práci jsem se podrobněji seznámil s problematikou minimalizování logických funkcí. Dále jsem si mohl otestovat své logické myšlení při implementaci této webové aplikace.

Nejdůležitější část při řešení této bakalářské práce, bylo testování aplikace, které trvalo cca 3 měsíce. Testování bylo zdlouhavé, neboť jsem musel otestovat co největší počet logických funkcí, které jsem si různě vymýšlel a poté jsem si je sám minimalizoval na papír a kontroloval aplikaci, zda minimalizaci prováděla správně. Jakmile nastávaly chyby, musel jsem si pečlivě zaznamenávat všechny parametry, které jsou použity v algoritmu a sledovat kde nastávají chybné výpočty. Největší problém byl v tom že, když jsem minimalizaci pro nějakou funkci opravil, objevili se mi například pro ostatních 5 logických funkcí opět chyby. Proto jsem se musel vracet a hledat kde tyto chyby nastávají. Jakmile jsem opravil chybu, bylo nezbytně nutné otestovat opět seznam logických funkcí a kontrolovat zda minimalizace probíhá správně. Tento postup jsem opakoval do doby, než byly všechny chyby odstraněny.

Aplikace může posloužit při výuce problematiky minimalizace logické funkce pomocí Karnaughovy mapy, kdy může uživatel sám minimalizovat a aplikace kontroluje, zda vytvořená smyčka je správně. Dále může být použita i nezávisle jenom jako nástroj pro minimalizaci logických funkcí pomocí Karnaughovy mapy.

Do budoucna by bylo možné aplikaci rozšířit o více vstupních proměnných, jelikož jsem se snažil, aby byla aplikace co nejvíce dynamická. Další možností rozšíření aplikace by mohlo být vykreslení schématu výsledné minimalizované funkce.

Literatura

[1] Z. Diviš, Z. Chmelíková a J. Zdrálek, Logické obvody, Ostrava: VŠB-Technická univerzita Ostrava, 2008.

[2] J. Klesl, Elektronika III, Číslicová technika, Praha: Ben-technická literatura, 2005.

[3] J.-M. Bernard, J. Hugon a R. L. Corvec, Od logických obvodů k mikroprocesorům, Praha: SNTL- Nakladatelství technické literatury, 1988.

[4] J. Kolenička, Logické systémy - Část I, Brno: Vysoké učení technické v Brně, 1976.

[5] J. Litschman, Logické obvody I., Ostrava: Vysoká škola báňská Ostrava, 1986.

Seznam příloh

K této bakalářské práci je přiloženo DVD medium, na kterém se nachází následující adresáře:

- **Silverlight_Minimalizace** – vytvořený projekt ve Visual Studiu 2010 ultimate
- **Aplikace** – spustitelná aplikace, která se otevře ve webovém prohlížeči
- **Bakalářská práce** – bakalářská práce ve formátu PDF/A
- **Manuál** – stručný uživatelský manuál k vytvořené aplikaci ve formátu PDF/A